SECURING THE USER'S WORK ENVIRONMENT

Nicholas P. Cardo
Sterling Software, Inc.
NASA Ames Research Center
Numerical Aerospace Simulation Facility
Moffett Field CA, 94035
Error! Bookmark not defined.

High performance computing at the Numerical Aerospace Simulation Facility at NASA Ames Research Center includes C90's, J90's and Origin2000's. Not only is it necessary to protect these systems from outside attacks, but also to provide a safe working environment on the systems. With the right tools, security anomalies in the user's work environment can be detected and corrected. Validating proper ownership of files against user's permissions will reduce the risk of inadvertent data compromise. The detection of extraneous directories and files hidden amongst user home directories is important for identifying potential compromises. The first runs of these utilities detected over 350,000 files with problems. With periodic scans, automated correction of problems takes only minutes. Tools for detecting these types of problems as well as their development techniques will be discussed with emphasis on consistency, portability and efficiency for both UNICOS and IRIX.

INTRODUCTION

Supercomputing plays an important role in today's research. Advanced research and time critical projects occupy the majority of the supercomputing cycles available. It is ever more important to secure this information and protect it from unauthorized access. The most widely noted problems are from external attacks on the system resulting in break-ins and possible theft of research material. Since computers are a shared resource, a second threat, internal unauthorized access, can be just as dangerous. For example, two groups of users may be competing against each other for the same project. Obtaining critical data for an opponent's design can provide an advantage. Protecting the research is a shared responsibility between the user and the system administrator.

A great effort has been spent on securing systems from external threats. However, gaining access to a system is only a part of the threat. Although systems may be relatively safe from external unauthorized access, they may be extremely vulnerable to internal threats. Securing information includes keeping it from unauthorized access within the user community. The magnitude of this problem is underestimated, as the true extent of this type of activity is unknown.

THE PROBLEMS

There are several problems that must be monitored on the system in order to protect the user's research from possible internal unauthorized access. In many cases, users are unaware that their actions have left their research data open for unauthorized access. The user's specialty is in their research, not in system administration and security. Providing the users with security related information about their files should be a service all centers provide.

File Ownership

Many computer centers operate on project allocations. That is, a user or group of users is granted a certain amount of computer time for their specific research. Computer time is typically tracked by an Account ID (ACID) or Project ID while file group ownership is based on the Group ID (GID). Grants typically have a specified life and must be renewed on some interval. At the time of renewing allocations, one of four things will happen to the user:

1. The user is disabled or removed
2. The user continues with no changes.
3. The user continues with additional projects.
4. The user continues with new projects.

Problems arise when users continue on with changes to their projects. While their project allocations are updated, what about the ownership of their files? For example, suppose a user of a project changes companies and is now working on a competing project with an allocation on the same computer. Unless group ownership is changed, the original company could unknowingly access research in this user's directory. Consequently, the extent of this problem is bi-directional. The result is that both companies could potentially gain access to each others work.

The solution to this problem is to periodically scan all files for all users and compare the group ownership of the file to the list of groups the user actually belongs to. Discrepancies should be reported and corrective action taken. Performing this type of scan at NAS identified over 350,000 files with problems. Continued scans helped to identify a procedural problem with the account conversion process. The problem turned out to be that ownerships were only being set on the first day of the new operational period. An account activated thereafter would retain the ownership from the previous operational period.

Another problem area is file ownership as it relates to filesystem quotas. Typically there are several users working on the same project and sharing data. In many cases they use a common directory for storing all their data files. Depending on how disk quotas are set up, this could be a problem. If the individuals of the same project are not on the same filesystem, then hiding files from their quota can be done by storing their data in another's directory on another filesystem. This type of problem can be overcome by using one quota file for all user filesystems. However, a good practice would be to set all the files in a users directory to be owned by that user. This also protects against unauthorized access problems.

Home Directory Ownership and Mode

It is important that the user's home directory maintains proper ownership as well as a mode setting which provides a safe operating environment. Regular comparisons of home directories against the user's properties can prevent potential problems. Again, in many cases, the users are unaware that they've done something to endanger their work. World writable directories can be devastating to a user's research. A world writable directory allows any user on the system to remove or replace files in that directory, without permission. In addition, bad group ownership can provide access to unauthorized users if group permissions are set. Group permissions allow any user within that group the ability to access that file or directory. Incorrect group ownership allows the wrong group of users to potentially have access to a file.

Home Directory Filesystems

Periodic comparisons of the home directories on the filesystems against the password file can identify two potential problems. The first problem is users without a valid login directory. The second problem is files and directories that should not exist in the same directory as users home directories. An easy place for someone to hide files and directories is amongst the home directories. Spotting problems by eye is difficult and unreliable due to the ability to name files or directories similar to valid login directory names.

.rhosts File

Checking the `$HOME/.rhosts` file for all users can help reduce the risk of external unauthorized access. Checking the ownership and mode of the file and the format of the contents is not sufficient. It is necessary to also check that the hosts listed in the file are valid. Any host that cannot be validated should be reported back to the user for corrective action. In many cases, entries for decommissioned hosts will be found in the file. Checking of this file can also identify violations of account sharing by comparing the users listed in the host against the user being checked.

Secure SHell

The Secure SHell (SSH) package adds more possibilities for incorrect configuration problems by the user. SSH provides the capability for a `$HOME/.shosts` file which operates similarly to the `$HOME/.rhosts` file. Similar checks need to be performed to the `.shosts` file as are done for `.rhosts` file.

SSH also allows the users to provide entries in their `$HOME/.ssh/authorized_keys` file for controlled access. This should be checked for account sharing and suspicious entries.

DEVELOPING APPLICATIONS

There are three design goals for developing security-related tools for UNICOS and IRIX. These are:

1. Speed
2. Flexability
3. Portability

Security tools should be fast in order to obtain accurate information. Long running tools may not provide a current accurate evaluation. In addition to accuracy, timeliness of results is an important factor with any security tool.

Security utilities need to have the flexibility to be adapted to evolving circumstances and configurations. Any utility developed will need to grow with the system. Security utilities typically have a clear and concise functionality. Having access to as much information as possible only enhances the utility's importance.

Portability

Any tool developed to perform a security function needs to be portable. SGI/Cray has already announced the phase out of traditional CRAY systems and UNICOS, with its replacement being the new scalable node architecture running a version of IRIX. As centers migrate from traditional CRAY systems running UNICOS towards the newer scalable node architectures such as the Origin2000 running IRIX, the necessity for portability becomes increasingly important. Eliminating the task of porting utilities between operating systems allows for more time to be devoted to the migration of customers without sacrificing security concerns.

Scripts versus Programs

Most security related tools that perform any type of directory tree traversal were scripts that utilized the `find` command. In order to properly evaluate potential file problems, the `find` command would have to execute a custom written program to analyze the file's attributes. For each file that the `find` command encounters, it would `stat` the file in order to evaluate the `find` command's options then `fork` and `exec` the custom written program which would again `stat` the file in order to evaluate the attributes. In early versions of UNIX, the find command provided the only means for traversing directory trees without developing tree traversal code.

A relatively unknown and unused function exists in UNICOS and IRIX that performs directory tree traversal. The function `ftw`, File Tree Walk, can be used by programs to traverse directory trees from within a program. This function calls a defined function within a program passing it the pathname, a `stat` structure, and identifying code information. A second form of `ftw` exists, `nftw`, which also performs the tree traversal but provides more options. On IRIX systems, both 32 and 64 bit functions exist.

The stat Structure

The `stat` structure provides a significant amount of information about files that can be utilized by security related utilities. Usefile fields in the `stat` structure take from `<sys/stat.h>` are:

`st_dev`	Device Number
`st_ino`	Inode Number
`st_mode`	File Mode
`st_nlink`	File Link Count
`st_uid`	User ID Ownership
`st_gid`	Group ID Ownership
`st_acid`	Account ID
`st_size`	File Size in Bytes
`st_atime`	Last Access Timestamp
`st_mtime`	Last Modified Timestamp
`st_ctime`	Last Inode Update Timestamp

Additional information is available for Multi-Level Security (MLS) and the CRAY Data Migration Facility (DMF).

It is important to understand the three time fields in the `stat` structure. The first, `st_atime`, provides a UNIX timestamp for the last access time of the file. The second, `st_mtime`, provides a UNIX timestamp for the last modify time of the file. The third, `st_ctime`, provides a UNIX timestamp for the last time the files attributes were changed. The last access and last modify times can be reset with the `utimes` function. However, the only way to alter `st_ctime` is modify the inode by opening the filesystem raw, locating the inode, and changing the information. Keep in mind that `st_ctime` is automatically set when any attribute of a file is changed. Changing the modification and last access times will cause `st_ctime` to be set to the time of the change. Understanding these three time fields is important in identifying problems. Suppose someone gained unauthorized access as root to a system. This individual then replaced the `passwd` program and reset the last

access and last modify times back to those of original programs. Unless the change time is examined, the file appears to be the same. The change time also provides an important clue while investigating this type of situation. The timestamp in `st_ctime` will help narrow down the timeframe in which the file changed.

Using ftw/nftw

Using ftw is a simple addition to a program. The function calls work similarly with the exception that `nftw` has an optional flags argument. The flags argument for nftw controls additional traversing criteria. There are 4 flags that can be set:

`FTW_PHYS`	Do not follow softlinks.
`FTW_MOUNT`	Do not cross mount points.
`FTW_DEPTH`	Subdirectories visited first.
`FTW_CHDIR`	Change to each directory before reading it.

The remaining arguments to `ftw`/`nftw` are the same. The first is a starting pathname to begin the search, the second is a function that `ftw`/`nftw` will call for each file, the third is the number of file descriptors to use.

`ftw`/`nftw` call a function passing it the pathname, a filled in `stat` structure for the file, flags identifying something about the file, and with `nftw` a base offset and level of the path being traversed.

One popular computer security technique is to maintain accurate listings of all `setuid` programs on the system. This would require traversing a filesystem and producing a listing of all `setuid` programs and some identifying information so that if they were to change, a warning message could be produced. Producing this information is an easy task with these functions.

```
main
{
   nftw(argv[optind], scantree, 32,
      FTW_PHYS|FTW_MOUNT);
}
int scantree(path,sb,code,fcode)
char     *path;
struct   stat sb;
int      code;
struct   FTW *fcode;

switch (code) {
   case FTW_DNR:
   case FTW_NS:
      fprintf(stderr,
         "no information for %s\n",path);
   case FTW_D:
   case FTW_DP:
   case FTW_SL:
      return(0);
   case FTW_F:
      break;
}
if (sb->st_mode & S_ISUID)
      printf("%s %d %d %o\n",path,
         sb->st_uid,sb->st_ctime
         sb->st_mode);
   return(0);
}
```

The `find` command could be used to accomplish this however it would require that a program be written to obtain the `uid` and `ctime`. This sample program shows how the functionality of the `find` command can be performed within a single program.

Reading the passwd File

Another useful capability is reading the `/etc/passwd` file from within a program. Using functions such as `getpwent()`, `getpwnam()`, or `getpwuid()` will return a filled-in structure containing all the information from the `passwd` file.

Similarly, the functions `getgrent()`, `getgrnam()`, or `getgrgid()` will return a filled-in structure containing information about a group. The possibility exists that the number of members of a group will cause multiple entries in the group file. Caution should be exercised to always continue reading the group file until the group name changes.

These functions provide quick easy ways to obtain information about users that can be used in any scanning program for comparisons. There are a set of routines explicitly written to extract information from the UNICOS UDB. However, the purpose of this paper is to discuss portable security utilities so the UDB functions will not be discussed.

USEFUL UTILITIES

Combining the fundamental functions already discussed into useful programs can produce fast and reliable security

related information. Here are just a few of the many utilities that can be developed.

Homeck

Making sure that all users have a valid login directory and that all directories on the home filesystems have valid users is useful for identifying problems. By using the information in `/etc/groups` and `/etc/passwd`, the proper login directory for every user can be identified along with proper ownership. Performing a comparison between the `passwd` file and the home filesystems will identify users with missing login directories, login directories with improper ownership, and directories on the home filesystems that don't correspond to valid users. Performing a check on the mode of the login directory can help enforce any established policies or at the minimum, warn the user of the potential problems.

Remotck

There are many ways to establish connections to systems and gain authorized access. It is also very easy to become complacent with these access methods and not perform the proper maintenance on access authorization files.

Using the `/etc/passwd` file for identifying the users login directory, checks on alternate login configuration files can be performed. Checking the `.rhosts` and `.shosts` files for valid entries and that each host exists or checking the `.ssh/authorized_keys` file can help find problems. In many cases, when systems are decommissioned, the entries are left in these files leaving potential vulnerabilities. Many users unknowingly leave themselves open to vulnerabilities.

Scandir

It is crucial to periodically validate all user files on the filesystems. This will help to eliminate vulnerabilities due to incorrect ownership or poor choice for the file mode. By using the information available in both the `/etc/group` and `/etc/passwd` file, each files ownership can be evaluated. Additional checking can be performed in the event that UNICOS ACIDs or IRIX Project Ids are used.

Sophisticated evaluation criteria can be added to this type of directory tree traversal utility. Problems such as `root` owned files in the user's directory or files without a valid owner in a user's directory could easily be detected.

Softlinks are another potential problem area. If softlinks cannot be resolved, that is the true file doesn't exist, data can be mistakenly or maliciously substituted. This is especially true when the softlink resolves to another user's directory. Dangling softlinks should be reported to the user so that corrective action can be taken.

Corrective Actions

Each utility should be constructed to allow for automated corrective actions. Corrective actions should include performing appropriate logging, making the necessary notifications, and correcting ownerships and file permissions.

Detecting the problem is only the first part of correcting problems. Taking the necessary action based on encountered problems is a necessity and can be time consuming. By building the corrective actions into the utility, problems can be speedily corrected and proper logging and notifications made.

SUMMARY

It is just as important to maintain a safe and secure environment for the user to work is as it is to protect the system from external vulnerabilities. As supercomputers become more powerful, there will be an increase in the demand for time on these systems. As more and more research is performed on these systems, the importance of protecting the users from internal vulnerabilities increases.

Both IRIX and UNICOS support the necessary library functions to allow for powerful and portable security utilities to be developed. What has worked in the past is not sufficient for what is needed now or in the future. It is time to take advantage of the advancements in library functions and system calls to develop new powerful and portable utilities.

No matter how safe a system is believed to be, it is vulnerable. Examining these types of internal vulnerabilities can be an eye opening experience. The responsibility for finding and correcting these problems falls on both the system/security administrator and the user. No system is 100% safe…

ACKNOWLEDGMENTS

This work was performed by Sterling Software at the Numerical Aerospace Simulation Facility at NASA Ames Research Center under NASA contract NAS2-13619.